

# WAVES Compiler/Generator: Optimum Performance & Transportability

Brad Roberts, John Stickley  
IKOS Systems, Inc.  
19050 Pruneridge Ave. Cupertino CA 95014  
email: brad@ikos.com Ph: (408) 366-8509

## 1.0 Abstract

The IEEE Waveform and Vector Exchange Specification (WAVES) standard 1029.1 [1] is a subset of VHDL specifically targeted at representing the stimulus that drives VHDL simulation models. The intent is to support users in the exchange of waveform information between different simulators and testers. By integrating a high performance, IEEE VHDL WAVES input-output capability into an accelerated VHDL mixed-level simulation environment, we can enhance WAVES ability to effectively support large, complex digital electronic system designs. The intent is to help the designer reduce design errors and shorten design schedules by using a standard stimulus format, with the advantage of mixed- and gate-level acceleration.

## 2.0 Introduction

The design industry needs electronic design automation tools and technologies that improve the design process, enabling designers to create working systems with reduced cost and risk. This means improving productivity throughout the entire design process, from capturing intent, to complete manufacturability. Using VHDL based design tools, with the added dimension of a WAVES stimulus capability, helps improve productivity and performance, thus allowing designers to quickly evaluate any problems affecting their designs.

This paper describes an approach that leverages existing C library functions [2], and accelerated stimulus capability, to implement a high-performance simulation solution for VHDL designers. The WAVES data can be loaded into the VHDL simulation kernel to provide improved WAVES code development and debugging. This means that the WAVES test bench can be developed and executed directly in the VHDL simulator, using the full debug capability of the VHDL environment. As the design grows, and portions of the design are stabilized, the WAVES files can be compiled into a stimulus format that directly feeds the simulator. After synthesis, when all of the design is at the gate level, the test bench is completely compiled and targeted to feed the hardware accelerator for maximum performance. This development flow eases design debug, provides validation at each step, thus improving simulation performance and throughput.

## 3.0 WAVES Interface Model

The WAVES input/output capability will have two main elements. The first element, the WAVES Compiler, will use an existing C interface to include models that implement WAVES stimulus data in a binary stimulus format (VStim). This format will be the input into both the VHDL simulation engine and the gate level simulation accelerator inside the VHDL mixed-level simulation environment. This will offer the choice of running the IEEE WAVES within the VHDL kernel, or in the more efficient, compiled binary format that will be directly injectable into the simulation environment at run time. This gives significant performance gains in simulations with long test vector data sets. We have seen gains as high as two orders of magnitude over simulations using equivalent standard behavioral VHDL test benches. These C models also have the ability to access simulation results on-the-fly and create custom stimulus output formats. These C models will allow us to implement the second element, the WAVES Generator, that will output simulation result vectors in a WAVES data set format.

This will improve the ease of WAVES creation, debug, and transport. These two mechanisms form the foundation for run time execution of IEEE WAVES simulation data.

#### 4.0 C Interface Overview

By using an extensive C procedural interface as part of the VHDL simulation environment, engineers and users can easily customize the simulation kernel. This C interface has been identified as the basis for the WAVES interface. All that is needed to communicate with the VHDL environment is a VHDL “wrapper” [3] and a “C” model. For each C model defined, one VHDL wrapper is required. This entity portion of the wrapper is basically a placeholder for the C model and contains the names and direction of the signals that interface with the rest of the VHDL design. The architecture simply contains the attribute that communicates to the simulation kernel that is implemented as a C model. When the wrapper, or entity-architecture pair, is analyzed inside the VHDL simulation environment, code is generated that links the signal ports to a single VHDL process. During elaboration (linking and loading of all VHDL design objects to the kernel) the shared object code for the model is linked with the simulation executable. See Figure 1 for illustration of C model inclusion.

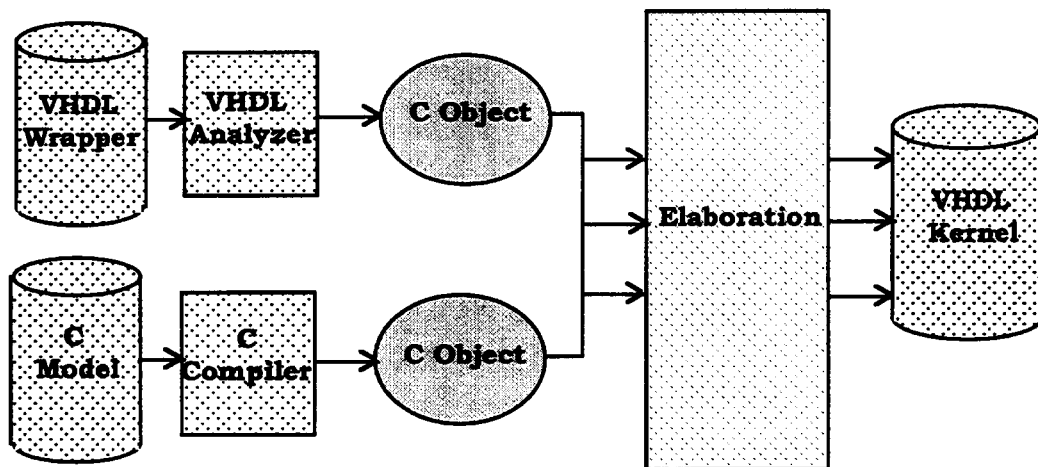


Figure 1 - C Model and VHDL Wrapper Inclusion

Within the VHDL wrapper, an architecture body is identified via an attribute named “foreign”. The value of this attribute is a string whose contents are used to identify the type of foreign architecture (e.g. a “C” language model), the elaboration of the process which implements the foreign architecture, and the object and archive files which comprise the architecture.

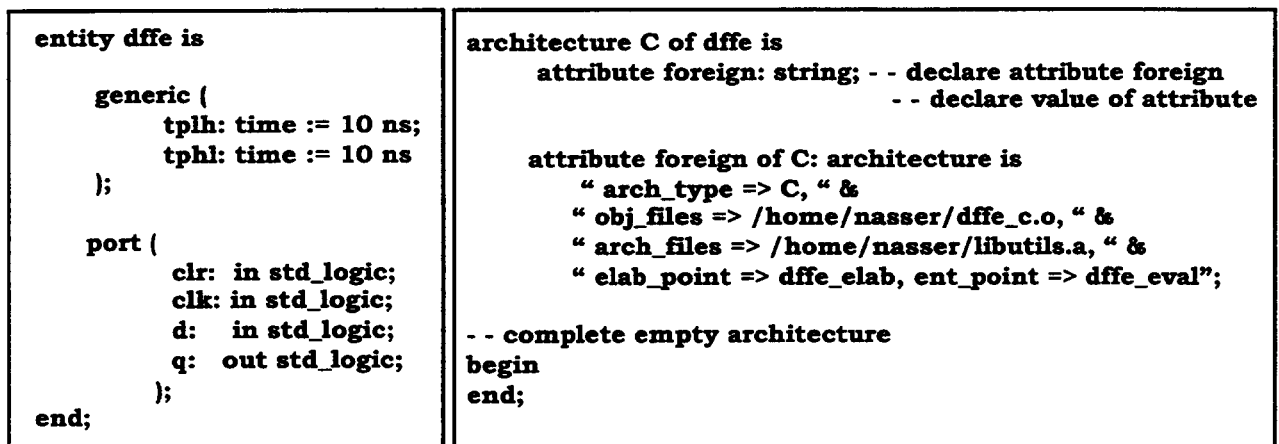


Figure 2 - Example of VHDL Wrapper “Entity-Architecture Pair”

The example shown in Figure 2 demonstrates both the entity and architecture body declaration of a foreign “C” model implementing a simple D flip-flop. With this C model concept in mind, we can see how it will be applied to the development of the WAVES Compiler/Generator. However, let’s first review the VStim Interface and see how we can leverage acceleration of compiled binary stimulus inside mixed-level VHDL simulations.

## 5.0 VStim Interface

Non-VHDL stimulus can be used to drive gate-level designs and can be used in conjunction with VHDL stimulus. These non-VHDL stimulus files are compiled and downloaded to the accelerator. As a result, simulation performance is enhanced when signals such as clocks, which have frequent events, are driven by these stimulus files. Additionally, non-VHDL stimulus can be used to indirectly drive VHDL modules in the design. In a mixed-level design (behavioral, RTL and gates) it is possible to use stimulus techniques other than the standard VHDL test bench. The VHDL models can be used to connect to the specified stimulus entities, which in turn drive the circuit under test. In other words, the gate-level stimulus is instantiated into a foreign (gate-level) block that in turn drives the unit under test (UUT) which is VHDL.

The VStim Interface [4] is the key to the performance for the WAVES Compiler/Generator. It’s primary function is to provide a mechanism for implementing vector stimulus inside the accelerator for the high-performance required for the complex designs of today. The two primary files that are created to connect to the accelerator are the flattened stimulus vector file (.vs) and the port list file (.pads). An example of the .pads file is illustrated below. It is a simple ASCII file that contains all the hierarchical connectivity information that works in conjunction with the binary .vs file that contains all the simulation events.

```

<# PORT LIST - NBS TRANSLATOR - Version 1.0 #>
<# #8 number of input pads
<# #0 number of output pads
<# Counted_Items I O R #>

##PADS 6 2 0

## BITS 50 18 0

<# INPUT TO THE HARDWARE ACCELERATOR #>

<# Instance_name Channel_name Pad_id Width Pad_direction Skew_delay #>

/uut/u0/1/JJ1 B1 1 1 I 0
/uut/u0/1/JJ3(5:0) B1 2 6 I 0
/uut/u0/1/JJ8 B1 3 1 I 0
/uut/u0/1/JJO1(12:3) B1 4 10 O 0
/uut/u0/1/JJO2(13:6) B1 5 8 O 0
/uut/u0/1/JJ12(5:0) B1 6 6 O 0
/uut/u0/1/JJ22(4:0) B1 7 5 O 0
/uut/u0/1/JJBIG(30:0) B1 8 31 O 0

<# OUTPUT FROM THE HARDWARE ACCELERATOR #>

<# Instance_name Channel_name Pad_id Width Pad_direction Skew_delay #>

##END_PORT_MAP

```

Figure 3 - Example of VStim “pads” file

## 6.0 WAVES Interface Architecture

We propose three new C models to enable the product to include WAVES Compiler /Generator capability and form the basis for the WAVES interface. These are the following:

- The VStim generator (WAVES compiler)
- The VStim reader/scheduler (WAVES compiler)
- The WAVES generator (WAVES generator)

We need to convert the WAVES data sets into an intermediate data format that is acceptable by the VHDL kernel and the hardware accelerator. We will use an existing analyzer to parse the WAVES data sets along with the VHDL model to create these intermediate data structures that will communicate with the VHDL kernel. The VHDL kernel itself is enhanced to convert these intermediate data structures into acceptable VStim format. The overall architecture for the WAVES Interface, containing all three C models, is illustrated in Figure 4.

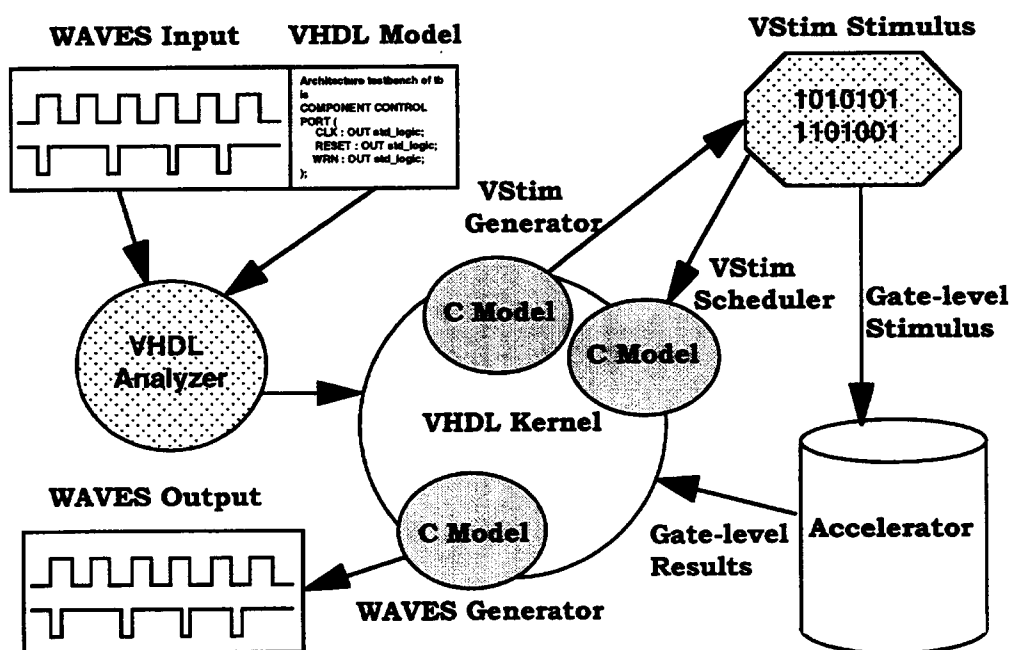
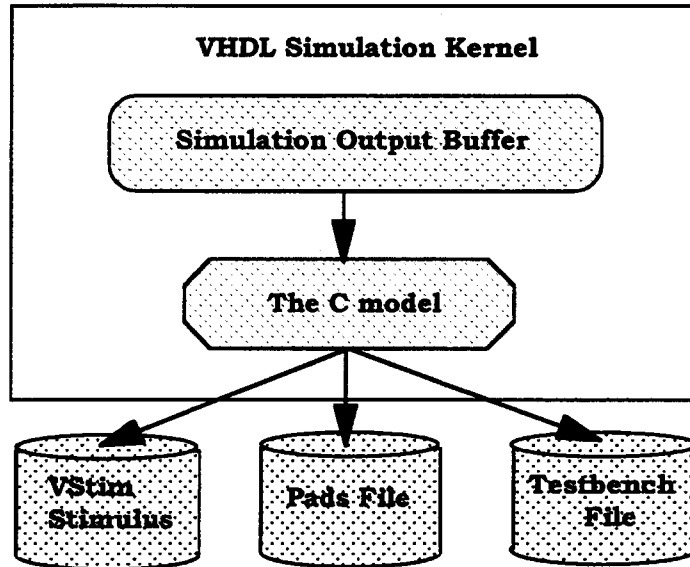


Figure 4 - WAVES Compiler/Generator Architecture

### 6.1 VStim Generator

The VStim Generator C model links to the VHDL simulation kernel via the VHDL wrapper. The WAVES data sets, along with the VHDL model, is instantiated in the empty architecture inside the VHDL wrapper, which is part of the test bench. The VStim file is created through analysis of the WAVES data sets and empty architecture, then linked to the VHDL kernel via the corresponding C model. This “captures” the results of the simulation which becomes the input file in VStim binary format. The VStim generator flushes the data from the kernel into VStim format whenever the kernel’s buffer gets full or the simulation is stopped. In other words, the results of the simulation are really the input stimulus that is being translated into VStim format. This VStim stimulus format is then used with the actual VHDL model to schedule events for the input signals for simulation. See Figure 4 for architecture of the VStim Generator inside the overall WAVES interface.

The VStim generator produces two more files, in addition to the VStim stimulus file, which are called the “pads file” and the “test bench file”. The pads file (example in Figure 6) contains information about the primary inputs and outputs of the Device Under Test (DUT) in a format required inside the VHDL simulation environment. It establishes connectivity between the VStim file and the VHDL kernel. The output data flow is illustrated in Figure 5.



**Figure 5 - Output Data Flow of VStim Generator**

The test bench file defines the foreign (non-VHDL) architectures of the VStim reader /scheduler, and the WAVES generator, and is used when the simulation is run with the accelerated WAVES data sets. An example of the test bench, with the added instantiations for the VStim reader and WAVES generator included, is illustrated in Figure 7. Based on the C model technology, it is required that the VStim Generator be instantiated within the original VHDL test bench. This can be done automatically by using an extension of the Test Bench Generation Tool [5] originally developed at Rome Laboratory in New York. The “TSTB” tool can be extended to produce a file which will be used by the VHDL simulator to specify signals being traced in the WAVES data sets.

```

<< PORT LIST - VStim Generator - Version 1.0 *>
<< #2 number of input pads
<< #2 number of output pads
<< Counted_Items I O R *>

###PADS 6 0 0

###BITS 4 0 0

<< INPUT TO THE HARDWARE ACCELERATOR*>

<< Instance_name Channel_name Pad_id Width Pad_direction Skew_delay *>
Clock B1 1 1 1 0
D B1 1 1 1 0

<< OUTPUT FROM THE HARDWARE ACCELERATOR*>

<< Instance_name Channel_name Pad_id Width Pad_direction Skew_delay *>
Q B1 1 1 0 0
Q_bar B1 1 1 0 0

###END_PORT_MAP
  
```

**Figure 6 - Pads output file from VStim Generator**

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY waves_std;
USE waves_std.waves_1164_utilities.all;

ENTITY test_bench IS END test_bench;

ARCHITECTURE d_flip_flop_test OF test_bench IS
  component d_flip_flop
    port ( clock: IN std_logic; D: IN std_logic;
          Q: OUT std_logic; Q_bar: OUT std_logic
        );
  end component;

  component vstim_gen end component;
  ...
begin
  .. signal definitions
  ...
  U1: d_flip_flop port map ( clock ...
                               );
  U2: vstim_reader;
  U3: waves_gen;
  ...
end d_flip_flop_test;

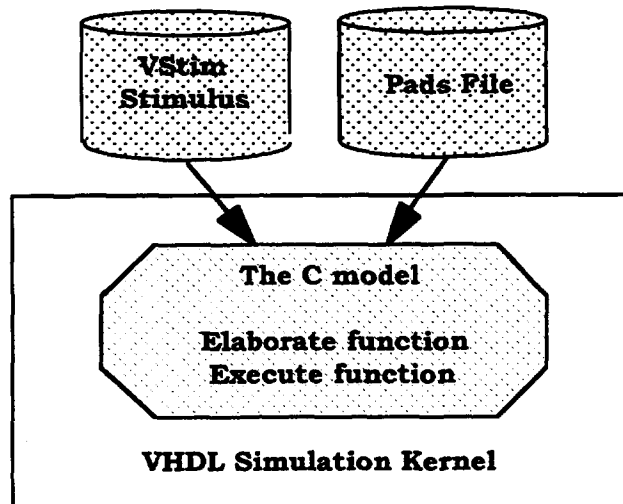
```

**Figure 7 - Test bench for VStim Generator**

In summary, the input files to the VStim Generator are nothing more than the IEEE WAVES data sets. This C model creates the necessary output files ( e.g. stimulus, pads, test bench) that are the inputs to the next C model, the VStim reader/scheduler.

### 6.2 VStim Reader/Scheduler

The VStim reader/scheduler is an enhancement to the VHDL kernel which reads and schedules events directly from the VStim stimulus file to maintain performance advantage. This is similar in structure to the VStim generator in that this new C model is linked into the kernel via a VHDL wrapper. The C model's function is to open the VStim stimulus file and schedule events for the input/inout signals while it reads the VStim stimulus records. This process continues until the reader reaches the end of the stimulus file. This process requires two input files; the VStim stimulus file and the pads file generated by the VStim generator.



**Figure 8 - Data Flow of VStim reader/scheduler**

The VStim reader/scheduler approach takes advantage of existing Foreign Kernel Interface (FKI) functions [6] which are part of the C Interface to read/schedule events in the simulation kernel. The FKI functions are a well defined set of C functions and are the foundation for third-party access to the VHDL simulation interface. The VStim reader/scheduler establishes the connection between the signals within the pads file and the VHDL model. The order is the following; first it opens the pads file to get each individual signal's name, width, and direction (input or output). It then uses an FKI function to get the "object\_id" for the signal name then uses this id to schedule the events within the C model. The C model contains two major parts; the elaborate and execute functions as shown in the flow inside Figure 8. The elaborate function obtains the "object\_id" of each individual signal, while the execute function uses these ids to access the ports inside the port list of the entity declaration. This execute function is called whenever any of the input, inout, or output ports to the C model change, or when the C model wakes up. This wake up happens upon expiration of time specified in a call to a specific FKI function (See Figure 10 for an example of the FKI functions).

### 6.3 WAVES Generator

The next C model is called the WAVES generator that is linked to the VHDL simulation kernel via the VHDL wrapper. The intent is to create WAVES data set output by extracting the data from the VHDL kernel whenever the kernel's buffer gets full or the simulation stops. The requirement is to generate WAVES data sets in Level 1 flat format [1]. The WAVES Level 1 format consists of; a WAVES header file, WAVES test\_pins package, WAVES generate procedure, WAVES test bench and the WAVES external file. The WAVES external file is an ASCII format that is consistent with the IEEE WAVES standard format. The order of signals matches the order in the test\_pins package file. The flow is shown in Figure 9 below.

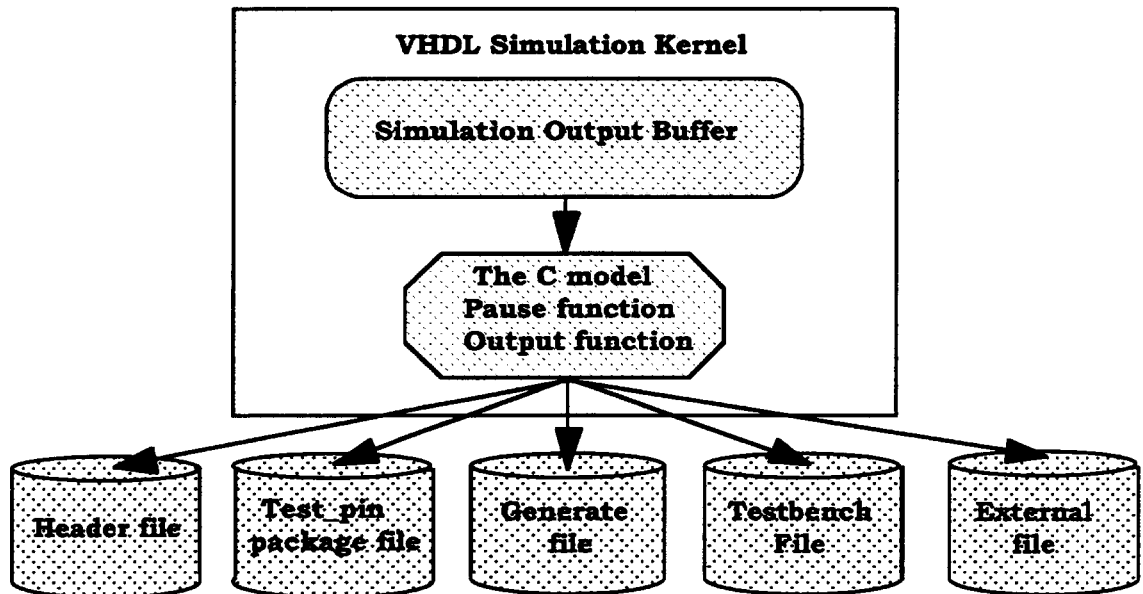


Figure 9 - Data Flow of WAVES data sets generator

This approach will take advantage of the existing Foreign Kernel Interface (FKI) and NNI functions to generate the results of the simulation in WAVES format at the highest possible speed; or as fast as the computer disk can upload. These NNI functions [7] are a subset of the FKI specifically developed for creating custom simulation output formats (e.g. WAVES) and have been optimized for fast output data handling. An example of these optimized functions for implementing the WAVES Generator is illustrated in Figure 10.

<u>WAVES Generator FKI Functions</u>	
double	fki_ReadGateResolutionUnits();
int	fki_ReadGateResolutionBase();
fki_Int_t	fki_Traces();
fki_String_t	fki_NextTrace(fki_Boolean_t, fki_Boolean_t);
void	fki_RegisterOutputHandler(*output_handler());
void	fki_RegisterPauseSimulationHandler(*pause_handler());
--	
<u>WAVES Generator NNI Functions</u>	
fki_Int_t	nni_GetTraceId(fki_String_t, fki_Int_t, fki_TypeClass_t, fki_TypeClass_t);
void	nni_Action(*user_process_event());
void	nni_EndAction(*user_process_event());
fki_Int_t	nni_GetOrder(fki_Int_t);
char*	nni_GetEntity();
char*	nni_GetArchitecture();
void	nni_ShowDeltaEvents(fki_Boolean_t);

**Figure 10 - FKI and NNI Functions for WAVES Generator**

Generating all the necessary files for WAVES data set format automatically and correct in structure, reduces the impact on the user and improves the design flow. The resulting WAVES output files can then be used to leverage the standard for future commercially available tools in the Computer Aided Test (CAT) community. Since all the output files for IEEE WAVES data set standards are well defined, it is an easier task to develop to this standard.

#### **6.4 Summary of WAVES Interface**

The C interface mechanism allows the designer to use one set of VStim stimulus data sets throughout the whole process of verification. This means these VStim data sets can be used to test functionality at the Behavioral/RTL levels and compare to runs using the same format, after synthesis, at the gate-level. Therefore, the entire design can be tested with the same set of stimulus throughout the whole design and test cycles. In addition to using the data sets throughout the entire design process, WAVES data sets and the corresponding VHDL model produce the output response based upon the stimulus applied to the simulation. This built-in test bench monitor helps ensure that the expected output matches the actual output response.

This same concept of monitoring “expected outputs” inside the VHDL simulation is just as powerful and important inside the gate- or mixed-level simulation. This interface will provide the same functionality via a gate-level comparator that is equivalent to the “window” and “window\_skew” functions inside the WAVES data sets package library. The signals to be compared are defined as expected output inside the VStim stimulus format. These expected outputs instruct the simulator to connect comparators for each signal in the stimulus file to the respective signal in the design. The simulation results are compared via a “stop on comparefail” command that includes user-defined “stobes” that are equivalent to the window and window\_skew functions. The result is notification of any mismatches of the expected outputs, at any level of implementation of the design.

Overall, the use of the WAVES Compiler/Generator achieves the original intent to support the IEEE WAVES standard as a complete stimulus format from original design through test. Figure 11 helps illustrate where the WAVES stimulus can be applied in a typical ASIC design flow.

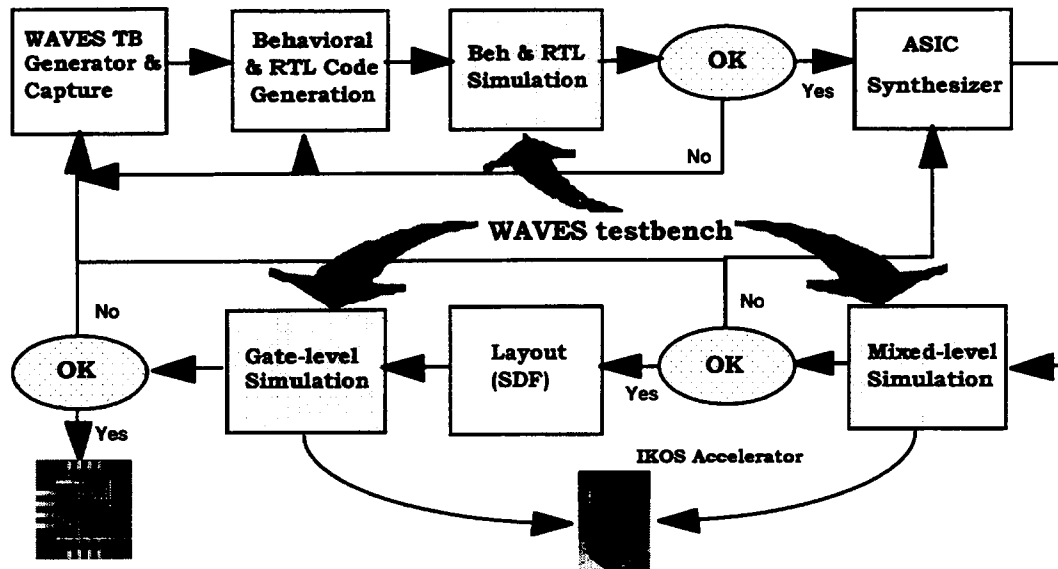


Figure 11 - WAVES Test Bench inside ASIC Design Flow

## 7.0 Test Bench Generation Tool

The WAVES Test Bench Generation Tool (TSTB) has been implemented by Rome Laboratory to aid VHDL users in developing their WAVES data sets and generation of the test bench for applying the stimulus and response to the VHDL models. The tool minimizes the impact on the user when first implementing WAVES inside their own design flow. The TSTB tool is a standard UNIX operational interface with the sole purpose of generating a VHDL compliant WAVES test bench along with the corresponding support files. This application contains a collection of integrated functions that automates tedious coding of these support files that are required in a WAVES data set.

The combination of the TSTB tool and the WAVES interface will ease the generation of WAVES test vector files necessary for today's complex ASIC and system level designs. After creating the definitions for all signals and waveforms, the designer can generate both IEEE WAVES compliant VHDL code for running behavioral and RTL simulations, and create an equivalent VStim compiled stimulus file by using the VStim generator. These two options satisfy the requirement for optimum performance, by running the compact binary VStim file on the accelerator, and transportability by generating compliant VHDL test benches. The intent is to package the TSTB tool set inside the WAVES Compiler/Generator interface to ease the development of the WAVES data set.

## 8.0 Conclusion

The use of a complete WAVES Compiler/Generator interface will help ease the VHDL designers transition to the WAVES standard. We can leverage the WAVES TSTB extension to help the VHDL designer minimize the work involved in creating WAVES test vector data files. The ease-of-use and ability to generate these WAVES test benches and supporting files inside a WAVES Compiler/Generator interface will help the designer create large regression tests needed in today's high-complexity, large gate count designs.

The key to this performance is the VStim interface. VStim evolved under the dual constraints of efficiency of representation, and performance of execution. It is a compact and efficient means to store waveform and vector data, yet is highly capable of effectively feeding the stimulus vectors into a hardware simulation accelerator. This also gives WAVES the ability to drive very large simulations with millions of gates in a co-simulation environment with large number of lines of VHDL code.

The option for creating either VHDL behavioral test benches to run in any commercially available VHDL simulator, along with the VStim binary stimulus for accelerated stimulus in mixed-level simulations on the IKOS accelerator, gives the designer the best of both worlds; high-performance and transportability.

## **9.0 References**

- [1]. "IEEE Standard for Waveform and Vector Exchange (WAVES)," Institute of Electrical and Electronics Engineers, IEEE Std 1029.1-1991
- [2]. "Voyager User's Guide and Reference Manual," Version 4.0, The C Interface: Reference Information, July 1996, pp. 1-71
- [3]. "Voyager User's Guide and Reference Manual," Version 4.0, Gate-Level Netlist Compilation - NACA Wrapper (Entity-Architecture) File, July 1996, pp. 457-460
- [4]. "Voyager User's Guide and Reference Manual," Version 4.0, Gate-Level Stimulus - Stimulus File List, July 1996, pp. 541-542
- [5]. Flynn, C.J., Hillman, R.G., Pronobis, M.T., "Test Insertion Without Being a Test Expert" Proceedings VHDL International User's Forum, Fall Conference, October 1995, pp. 10.5 -10.7
- [6]. "Voyager User's Guide and Reference Manual," Version 4.0, The C Interface: Reference Information - FKI Tracing Calls, July 1996, pp. 60-67
- [7]. "Voyager User's Guide and Reference Manual," Version 4.0, The C Interface: Reference Information - NNI Calls for Creating Custom Output Format, July 1996, pp. 68-71